Schwarz–Schur Involution: Lightspeed Differentiable Sparse Linear Solvers

Yu Wang, S. Mazdak Abulnaga, Yaël Balbastre, Bruce Fischl

MGH 1811 · HARVARD MEDICAL SCHOOL · MIT · SONY · Martinos Center for Biomedical Imaging · Sony Interactive Entertainment · UCL

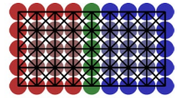
International Conference on Machine Learning 2025

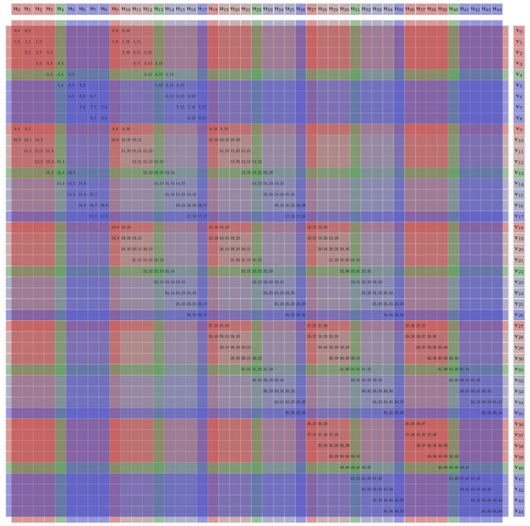
Problem: sparse solve aka generalized deconvolution

Consider an $H \times W$ image with $n = HW$ pixels, the matrix $\mathrm{A} \in \mathbb{R}^{n \times n}$ has the Laplacian-like sparsity: $\mathrm{A}_{ij} \neq 0$ only if pixels i and j are adjacent in the image. E.g., for a 9×5 image, the matrix A looks like:

Each pixel corresponds to a row/column in the matrix A

$\mathrm{A}_{ij} \neq 0$ only if pixels i and j connected by an edge





Solving the sparse linear system $\mathrm{A}^{-1}\mathrm{b}$: equivalent to de-convoluting an image b with a spatially varying kernel A (rows of sparse A store the kernel)

- consider a 3×3 convolution kernel that is spatially varying:

$$a^{(x,y)} := \begin{bmatrix} a^{(x,y)}(-1,-1) & a^{(x,y)}(-1,0) & a^{(x,y)}(-1,1) \\ a^{(x,y)}(0,-1) & a^{(x,y)}(0,0) & a^{(x,y)}(0,1) \\ a^{(x,y)}(1,-1) & a^{(x,y)}(1,0) & a^{(x,y)}(1,1) \end{bmatrix}.$$

$$v(x,y) \leftarrow \sum\sum_{\delta_x,\delta_y \in \{-1,0,1\}} a^{(x,y)}(\delta_x,\delta_y) u(x+\delta_x, y+\delta_y)$$

- $\mathrm{v} \leftarrow \mathrm{A}\,\mathrm{u}$: convolution (generalized to spatially varying kernels)
- $\mathrm{u} \leftarrow \mathrm{A}^{-1}\mathrm{v}$: deconvolution (in the exact, generalized sense)

Finite element method / numerical PDEs in a nutshell:

- elliptic PDE $\nabla \cdot [\mathrm{C}(\mathrm{x})\nabla u(\mathrm{x})] = 0$ with $\mathrm{n}(\mathrm{x})^{\mathsf{T}}[\mathrm{C}(\mathrm{x})\nabla u(\mathrm{x})] = g(\mathrm{x}), \forall \mathrm{x} \in \partial\Omega$ i.e., Neumann boundary condition, amounts to $\mathrm{A} \leftarrow \mathrm{L}$, where $\mathrm{L} \coloneqq \mathrm{G}^{\mathsf{T}}\mathrm{C}\mathrm{G}$;
- the parabolic PDE amounts to setting $\mathrm{A} \leftarrow \mathrm{t} \cdot \mathrm{L} + \mathrm{M}$
- the Helmholtz equation amounts to $\mathrm{A} \leftarrow \mathrm{L} - \kappa^2 \cdot \mathrm{M}$

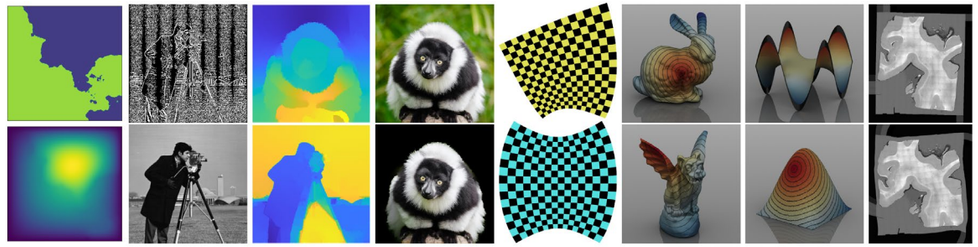
  M: mass matrix, G: gradient; plays roles like the identity, adjacency matrices

Linear solvers realize the *coefficient-to-solution map* $\mathrm{C} \mapsto [\mathrm{G}^{\mathsf{T}}\mathrm{C}\mathrm{G}]^{-1}\mathrm{b}$

Summary:

- **Sparse linear system $\mathrm{x} = \mathrm{A}^{-1}\mathrm{b}$ solved up to 1000X faster**
- the first direct solver that can run interactively. $\mathrm{x} = \mathrm{A}^{-1}\mathrm{b}$ can be: a PDE/FEM solver, generalized deconvolution, exact Newton's solver, deformer, geometry solver, physics solver, spectral solver, +more

Application: AI4PDE, scientific computing, computer vision, +more



Observation: sparse solvers too slow—unnecessarily

A 4097×4097 image, *n*=16785409 pixels, divided into a 1024×1024 array of overlapping 5×5 patches

- SciPy: 20 minutes to solve the linear system
- 0.008 seconds to invert one million 9×9 matrices removing most pixels (9/16≈56%) as a Gaussian elimination step

```
# alpha.shape is (1024,1024,9,9)
torch.linalg.inv(alpha) #0.008 sec.
# A: sparse matrix
scipy.sparse.linalg.spsolve(A,b) #1200 sec.
```

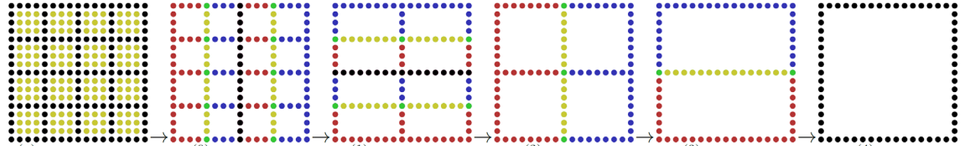
Approach: tensorize and parallelize the sparse linear solve to transfer GPU's dense BLAS capacity to sparse problems

- convert the observation into algorithms: tensorized representations of batched reduced systems
- divide A into tensor $\boldsymbol{\alpha}^{(*)} \in \mathbb{R}^{\frac{W}{4} \times \frac{H}{4} \times 25 \times 25}$, divide b into tensor $\boldsymbol{\beta}^{(*)} \in \mathbb{R}^{\frac{W}{4} \times \frac{H}{4} \times 25 \times 1}$

Algorithms: "involuting" tensors $(\boldsymbol{\alpha}, \boldsymbol{\beta})$

—a callable module users *need not* to implement by themselves

- recursively collapse sub-domains by merging sub-systems: $\mathrm{P} = \boldsymbol{\alpha}[i,\ j,\ :,\ :]$, $\mathrm{Q} = \boldsymbol{\alpha}[i+1,\ j,\ :,\ :]$
- $\boldsymbol{\alpha}^{(k)}[i,\ j,\ :,\ :]$ represents a (generalized) discrete *Dirichlet-to-Neumann* operator

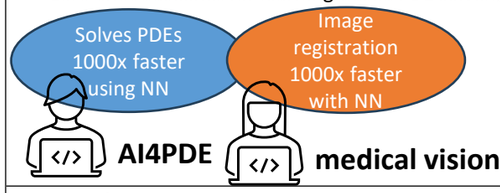


$\boldsymbol{\alpha}^{(*)}: (4,4,25,25) \rightarrow \boldsymbol{\alpha}^{(0)}: (4,4,16,16) \rightarrow \boldsymbol{\alpha}^{(1)}: (2,4,24,24) \rightarrow \boldsymbol{\alpha}^{(2)}: (2,2,32,32) \rightarrow \boldsymbol{\alpha}^{(3)}: (1,2,48,48) \rightarrow \boldsymbol{\alpha}^{(4)}: (1,1,64,64).$

$\boldsymbol{\beta}^{(*)}: (4,4,25,1) \rightarrow \boldsymbol{\beta}^{(0)}: (4,4,16,1) \rightarrow \boldsymbol{\beta}^{(1)}: (2,4,24,1) \rightarrow \boldsymbol{\beta}^{(2)}: (2,2,32,1) \rightarrow \boldsymbol{\beta}^{(3)}: (1,2,48,1) \rightarrow \boldsymbol{\beta}^{(4)}: (1,1,64,1).$

Broadly Impacted Areas:

- generalized deconvolution of spatially varying kernels for image processing, vision
- numerical optimization: exact Newton solvers made tractable on image domains
- solver/optimizer layers embedded in neural nets, (physics/geometry) solver-in-the-loop
- geometric deep learning & algorithms, shape/deformation representation
- eigenbases for spectral neural networks, spectral clustering (Shi & Malik)
- identify/reduce gaps between AI & conventional methods
- zero-shot baselines for learning-based PDE solvers, scientific ML, AI4PDE



Ours: these problems can be solved 1000x faster ***without*** **using any learning / neural nets** (NN)—**only GPUs are enough.**

Results

Even using a naïve prototype:

- 60~1000x faster than SciPy. 40~170x faster than CUDA (cuDSS, cuSparse)
- taking our method 10.9 ms (resp. 220 ms) to solve a Laplacian system (Dirichlet) on an image of 513×513 (resp. 2561×2561)

Example	CUDA	SciPy	ours	speedup
2561^2	36926	253318	220.1	**168X 1151X**
2049^2	21354	143100	158.5	**135X 903X**
1025^2	4710	16512	36.45	**129X 453X**
513^2	1036	2051	10.90	**95.0X 188X**
257^2	234	355	5.82	**40.2X 61.0X**

Discussion: why not iterative solvers?

- cannot reuse: must restart when sequentially solve multiple $(\mathrm{A}, \mathrm{b}_1), (\mathrm{A}, \mathrm{b}_2), \ldots, (\mathrm{A}, \mathrm{b}_k)$
- each method & parameter setting only applies to a small range of A
- struggle to deconvolute indefinite kernels such as stencils discretizing Helmholtz PDEs
- unpredictable runtime, unreliable without case-by-case user intervention
- parameters, convergence time, preconditioning (and its parameters): all depend on A

Schur step: merge sub-systems (P, p), (Q, q) into (D, d)

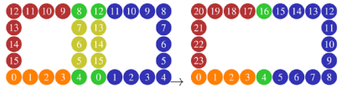


Figure 4. Schur step collapses subdomains P and Q into D.

$$\begin{bmatrix} \mathrm{P}_{\alpha\alpha} & \mathrm{P}_{\alpha\beta} & \mathrm{P}_{\alpha\gamma} & \mathrm{P}_{\alpha\delta} & \mathrm{P}_{\alpha\epsilon} \\ \mathrm{P}_{\beta\alpha} & \mathrm{P}_{\beta\beta} & \mathrm{P}_{\beta\gamma} & \mathrm{P}_{\beta\delta} & \mathrm{P}_{\beta\epsilon} \\ \mathrm{P}_{\gamma\alpha} & \mathrm{P}_{\gamma\beta} & \mathrm{P}_{\gamma\gamma} & \mathrm{P}_{\gamma\delta} & \mathrm{P}_{\gamma\epsilon} \\ \mathrm{P}_{\delta\alpha} & \mathrm{P}_{\delta\beta} & \mathrm{P}_{\delta\gamma} & \mathrm{P}_{\delta\delta} & \mathrm{P}_{\delta\epsilon} \\ \mathrm{P}_{\epsilon\alpha} & \mathrm{P}_{\epsilon\beta} & \mathrm{P}_{\epsilon\gamma} & \mathrm{P}_{\epsilon\delta} & \mathrm{P}_{\epsilon\epsilon} \end{bmatrix} := \mathrm{P}, \quad \begin{bmatrix} \mathrm{p}_\alpha \\ \mathrm{p}_\beta \\ \mathrm{p}_\gamma \\ \mathrm{p}_\delta \\ \mathrm{p}_\epsilon \end{bmatrix} := \mathrm{p},$$

$$\begin{bmatrix} \mathrm{Q}_{\kappa\kappa} & \mathrm{Q}_{\kappa\lambda} & \mathrm{Q}_{\kappa\mu} & \mathrm{Q}_{\kappa\nu} \\ \mathrm{Q}_{\lambda\kappa} & \mathrm{Q}_{\lambda\lambda} & \mathrm{Q}_{\lambda\mu} & \mathrm{Q}_{\lambda\nu} \\ \mathrm{Q}_{\mu\kappa} & \mathrm{Q}_{\mu\lambda} & \mathrm{Q}_{\mu\mu} & \mathrm{Q}_{\mu\nu} \\ \mathrm{Q}_{\nu\kappa} & \mathrm{Q}_{\nu\lambda} & \mathrm{Q}_{\nu\mu} & \mathrm{Q}_{\nu\nu} \end{bmatrix} := \mathrm{Q}, \quad \begin{bmatrix} \mathrm{q}_\kappa \\ \mathrm{q}_\lambda \\ \mathrm{q}_\mu \\ \mathrm{q}_\nu \end{bmatrix} := \mathrm{q}.$$

$$\mathrm{X} := \begin{bmatrix} \mathrm{P}_{\alpha\alpha} & \mathrm{P}_{\alpha\beta} & 0_{\alpha\lambda} & \mathrm{P}_{\alpha\delta} & \mathrm{P}_{\alpha\epsilon} \\ \mathrm{P}_{\beta\alpha} & \mathrm{P}_{\beta\beta} + \mathrm{Q}_{\kappa\kappa} & \mathrm{Q}_{\kappa\lambda} & \mathrm{P}_{\beta\delta} + \mathrm{Q}_{\kappa\mu} & \mathrm{P}_{\beta\epsilon} \\ 0_{\lambda\alpha} & \mathrm{Q}_{\lambda\kappa} & \mathrm{Q}_{\lambda\lambda} & \mathrm{Q}_{\lambda\mu} & 0_{\lambda\epsilon} \\ \mathrm{P}_{\delta\alpha} & \mathrm{P}_{\delta\beta} + \mathrm{Q}_{\mu\kappa} & \mathrm{Q}_{\mu\lambda} & \mathrm{P}_{\delta\delta} + \mathrm{Q}_{\mu\mu} & \mathrm{P}_{\delta\epsilon} \\ \mathrm{P}_{\epsilon\alpha} & \mathrm{P}_{\epsilon\beta} & 0_{\epsilon\lambda} & \mathrm{P}_{\epsilon\delta} & \mathrm{P}_{\epsilon\epsilon} \end{bmatrix}$$

$$\mathrm{Y} := \begin{bmatrix} \mathrm{P}_{\alpha\gamma} \\ \mathrm{P}_{\beta\gamma} + \mathrm{Q}_{\kappa\nu}\mathrm{J} \\ \mathrm{Q}_{\lambda\nu}\mathrm{J} \\ \mathrm{P}_{\delta\gamma} + \mathrm{Q}_{\mu\nu}\mathrm{J} \\ \mathrm{P}_{\epsilon\gamma} \end{bmatrix} \quad \mathrm{y} := \begin{bmatrix} \mathrm{p}_\alpha \\ \mathrm{p}_\beta + \mathrm{q}_\kappa \\ \mathrm{q}_\lambda \\ \mathrm{p}_\delta + \mathrm{q}_\mu \\ \mathrm{p}_\epsilon \end{bmatrix} \quad \mathrm{x} := \begin{bmatrix} \mathrm{u}_\alpha \\ \mathrm{u}_\beta \\ \mathrm{u}_\lambda \\ \mathrm{u}_\delta \\ \mathrm{u}_\epsilon \end{bmatrix}$$

$$\mathrm{Z} := \begin{bmatrix} \mathrm{P}_{\gamma\alpha} & \mathrm{P}_{\gamma\beta} + \mathrm{J}^{\mathsf{T}}\mathrm{Q}_{\nu\kappa} & \mathrm{J}^{\mathsf{T}}\mathrm{Q}_{\nu\lambda} & \mathrm{P}_{\gamma\delta} + \mathrm{J}^{\mathsf{T}}\mathrm{Q}_{\nu\mu} & \mathrm{P}_{\gamma\epsilon} \end{bmatrix}$$

$$\mathrm{W} := \begin{bmatrix} \mathrm{P}_{\gamma\gamma} + \mathrm{J}^{\mathsf{T}}\mathrm{Q}_{\nu\nu}\mathrm{J} \end{bmatrix} \quad \mathrm{w} := \begin{bmatrix} \mathrm{p}_\gamma + \mathrm{J}^{\mathsf{T}}\mathrm{q}_\nu \end{bmatrix}$$

$$\mathrm{D} := (\mathrm{X} - \mathrm{Y}\mathrm{W}^{-1}\mathrm{Z}) \quad \mathrm{d} := \mathrm{y} - \mathrm{Y}\mathrm{W}^{-1}\mathrm{w}$$

Acknowledgement: Justin Solomon, Mike Taylor.